**Detecting and preventing abuse on LinkedIn using isolation forests**

The Anti-Abuse AI Team at LinkedIn creates, deploys, and maintains models that detect and prevent various types of abuse, including the creation of fake accounts, member profile scraping, automated spam, and account takeovers. There are several unique challenges we face when using machine learning to prevent abuse on a large professional network, including:

* **Labels:** few/poor “ground truth” labels, or sometimes none at all
* **Adversarial:**attackers are quick to adapt and evolve
* **Unbalanced:**the abusive traffic is a very small fraction of total traffic
* **Many surfaces:**there are many dynamically-changing heterogeneous surfaces to monitor

To help solve these challenges, we’ve created a Spark/Scala implementation of the Isolation Forest unsupervised outlier detection algorithm. Today, we are announcing that our Isolation Forest library is now open sourced and [available on GitHub](https://github.com/linkedin/isolation-forest).

In this post, we provide a technical overview of the Isolation Forest algorithm. Then, we describe our Spark/Scala Isolation Forest library, as well as describing the above challenges in more detail and outlining how Isolation Forests provide a solution. Finally, we share some concrete potential use cases for this algorithm.

Isolation Forests

The Isolation Forest algorithm was [first proposed in 2008 by Liu et al.](https://doi.org/10.1109/ICDM.2008.17) It is a type of unsupervised outlier detection that leverages the fact that outliers are “few and different,” meaning that they are fewer in number and have unusual feature values compared to the inlier class. Liu et al.’s innovation was to use a randomly-generated binary tree structure to non-parametrically capture the multi-dimensional feature distribution of the training dataset.

Each isolation tree is created using the following steps:

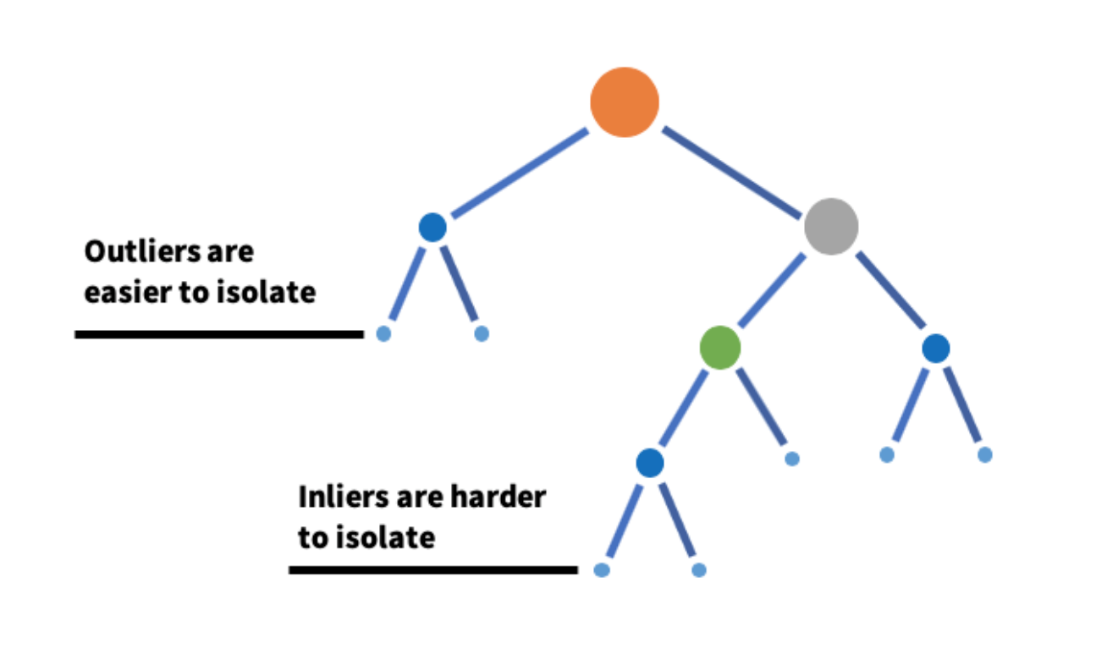
    1. Randomly sample *N* instances from your training dataset.

At each node:

    2. Randomly choose a feature to split upon.

    3. Randomly choose a split value from a uniform distribution spanning from the minimum value to the maximum value of the feature chosen in Step 2.

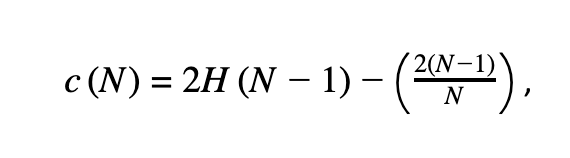
Steps 2 and 3 are repeated recursively until, in principle, all *N* instances from your sample are “isolated” in leaf nodes of your isolation tree—one training instance per leaf node. In practice, we don’t need to build the tree so deeply and can apply a height limit.

* 

*An example isolation tree*

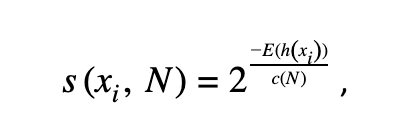
The intuition is that outliers—due to being few and different—are easier to isolate in leaf nodes and thus require fewer random splits to achieve this, on average. The result is a shorter expected path length from the root node to the leaf node for outlier data points. The outlier score for a particular instance is a function of the path length from the root node to the leaf node and the total number of training instances used to build the tree.

If a height limit is applied when building the tree, some leaf nodes will end up with more training instances than others. This is useful additional information that should be incorporated into the outlier score. The average depth for an unsuccessful search in a binary search tree created with *N* instances is given by

* 

where *H*(*i*) ≈ ln(*i*) + 0.5772156649. Due to the similar structure of binary search trees and isolation trees, the value *c*(*N*) provides the average depth of an isolation tree created using *N* training instances. For leaf nodes containing *M* > 1 training instances, we can add *c*(*M*) to the measured path length from the root to the leaf node to account for the number of instances terminating in the leaf node; this sum yields the effective path length for a particular instance, *h*(*xi*).

We can train an ensemble of isolation trees, an Isolation Forest, and average across their output to reduce the variance of the model. Once an Isolation Forest model is trained, the outlier score for an instance *xi* is given by

* 

where *E*(*h*(*xi*)) is the effective path length for that instance, *h*(*xi*), averaged across all trees in the ensemble, and *c*(*N*) is the expected depth of an isolation tree given *N* training instances discussed previously. This un-calibrated score, *s*(*xi*, *N*), ranges from 0 to 1. Higher scores are more outlier-like.

Isolation Forest Spark/Scala library

We created a Spark/Scala implementation of the Isolation Forest algorithm for use at LinkedIn. Our implementation supports distributed training and scoring using Spark data structures. We inherit Spark ML’s Estimator and Model classes in order to take advantage of Spark ML machinery such as Pipelines. Our implementation supports model persistence on Hadoop Distributed File System (HDFS).

Here is an example demonstrating how to import the library, create a new IsolationForest instance, set the model hyperparameters, train the model, and then score the training data. In this example, data is a Spark DataFrame with a column named features that contains a Vector of the attributes to use for training. In this example, the DataFrame data also has a labels column; it is not used in the training process, but could be useful for model evaluation.

*How to train and score data in Scala using our Isolation Forest library*

The output DataFrame, dataWithScores, is identical to the input data DataFrame but has two additional result columns appended with their names set via model parameters; in this example, these are named predictedLabel and outlierScore.

Why use Isolation Forests?

We chose the [Isolation Forest algorithm](https://doi.org/10.1109/ICDM.2008.17) for multiple reasons. Isolation Forests are a [top-performing unsupervised outlier detection algorithm](https://doi.org/10.1016/j.patcog.2017.09.037). Isolation Forests are also scalable, as their computational and memory requirements are low compared to common alternatives. There are fewer assumptions (e.g., non-parametric, no need for a distance metric) than other candidate algorithms. Finally, Isolation Forests are actively used in academia and industry, which allows us to leverage best practices and new developments shared by others in the field.

**Labels**

For some types of abuse, such as spam, it is possible to have a scalable review process where humans label training examples as spam or not spam. There are other types of abuse, such as scraping, where this kind of scalable human labeling is much more difficult, or impossible. Often, the labels you are able to obtain for training and evaluation are fuzzy; the precision may be less than ideal and there may be poor recall for some types of abusive behavior. Unsupervised techniques such as Isolation Forests are designed for problems with few or no labels, so they help to circumnavigate these label-based challenges.

**Adversarial**

The lack of good labels for training is further complicated by the fact that the problem is adversarial. Bad actors are often quick to adapt and evolve in sophisticated ways. Even if we are able to obtain some labels for abusive behavior identified today, the labels may not be representative of what abusive activity looks like tomorrow.

Isolation Forests can be used to overcome this challenge. As long as new abusive behavior is located in a different region of the feature space than normal, organic user behavior, we often can detect it using outlier detection—even if we did not have labeled examples when the model was trained.

**Unbalanced**

Abusive behavior is a very small fraction of all member activity on LinkedIn. This is a natural use case for outlier detection, where the outlier class is expected to be fewer in number compared to the inlier class.

**Many surfaces**

The Anti-Abuse Team at LinkedIn detects and prevents a wide variety of abuse across a diverse set of product surfaces. Our models often use features calculated using events from tracking infrastructure owned by other teams. This is a heterogeneous, dynamic environment that requires the use of a generalizable modeling strategy that supports easy retraining as the infrastructure changes underneath our models. Isolation Forests are easy to retrain if feature distributions shift, which helps to satisfy this requirement.

Potential uses for Isolation Forests

There are many uses for unsupervised outlier detection in the abuse detection domain and other related areas at large internet companies, including:

* **Automation detection:**Identify abusive accounts that are using automation to scrape data, send spam, or generate fake engagement
* **Advanced persistent threats:**Identify and prioritize sophisticated fake accounts for review by human experts
* [**Insider threats/intrusion detection**](http://lup.lub.lu.se/student-papers/record/8952203)**:**Detect compromised employee machines via anomalous network traffic
* **ML health assurance:**Automatically detect anomalous feature values and shifts in feature distributions
* [**Account takeover**](https://jverbus.github.io/data%20science%20projects/2016/10/07/insight-castle-compromised-account-detection)**:** Increase recall for account takeover detection
* **Alerting on time-series data:**Find anomalies in multi-dimensional time-series data
* [**Payment fraud**](https://arxiv.org/abs/1811.02196)**:** Flag suspicious payments to prevent fraud
* [**Data center monitoring**](https://doi.org/doi:10.1002/cpe.4169)**:**Automatically detect anomalies in data center infrastructure

The Isolation Forest library is now open source

As a result of the successful use of the library across multiple abuse verticals at LinkedIn, we have released it as open source software. You can find our Isolation Forest library [on GitHub](https://github.com/linkedin/isolation-forest).